Project report - ArduPilot

Sina Khoshfetrat Pakazad, Daniel Ankelhed

Division of Automatic Control

E-mail: sina.kh.pa@isy.liu.se, ankelhed@isy.liu.se

2nd March 2011

Report no.: LiTH-ISY-R-2996

Abstract

Ardu software, for autopilot functionality has been used for several years for many different UAV platforms. The aim of this project is to use the same functionality for the plane platform existing in the department. Considering the mismatch of the existing platform with the previously used ones for Ardu autopilots it is of great importance to alter the software and hardware provided by Ardu in order to be able to fly the existing UAV.

Keywords: UAV, autopilot, IMU, ArduPilot, ArduIMU.

Project report - ArduPilot

Version 0.1

Sina Khoshfetrat Pakazad, Daniel Ankelhed

2011-03-02

1 Project description

Ardu software, [1], for autopilot functionality has been used for several years for many different UAV platforms. The aim of this project is to use the same functionality for the plane platform existing in the department. Considering the mismatch of the existing platform with the previously used ones for Ardu autopilots it is of great importance to alter the software and hardware provided by Ardu in order to be able to fly the existing UAV.

1.1 Goals

- Match software and hardware of the ArduPilot to the aircraft. Considering the fact that the existing platform is self-stabilizing and that we only have control over the elevator and rudder it is important to disable the stabilizing function of the ArduPilot and alter the existing models with a more consistent model of the UAV platform.
- Modify the software and perform open loop and closed loop tests for the purpose of smart tuning of controller parameters.
- Use the functionality of way-point tracking by only using the GPS measurements.

1.2 Bonus goals

- Examine the possibility of mounting a camera on the UAV.
- Examine the possibility and challenges of communication and collaboration between several UAVs towards completion of a desired task.

1.3 Extra goals

After the start of the project, another goal was added.

- Connect an IMU (Inertial Measurement Unit) as an extra sensor and use it to improve the control of the airplane. Adapt the software if necessary.

2 Equipment setup

In this section, we briefly describe the equipment that is used in the project.

2.1 The airplane

In this section we describe the airplane and what the differences are compared to the standard Easystar platform.

2.1.1 A brief description

The plane that we use for the task is shown in Figure 1. The plane in made of a very light material and thus weighs very little, but on the other hand that makes it extra sensitive to wind gusts. The plane has elevator and rudder control surfaces but no ailerons. It is self-stabilizing around the roll axis (X-axis). This fact should make it possible to control without any angle measurements.

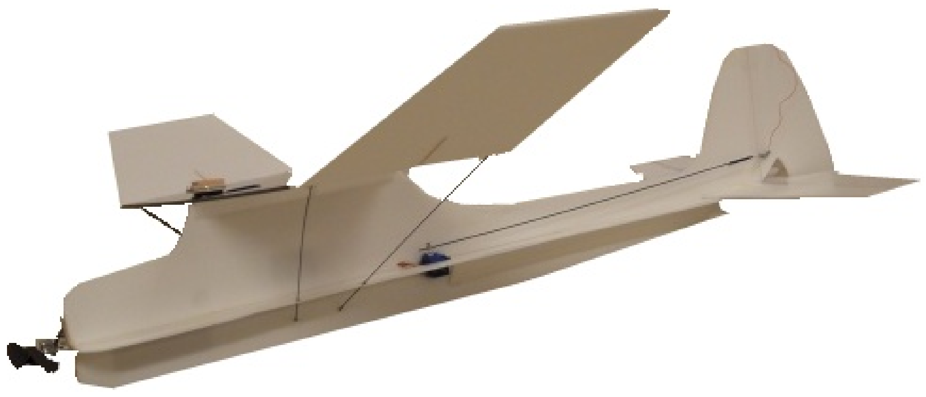


Figure 1: The plane that we use in the project. The control surfaces are rudder and elevator only.

2.1.2 Differences with the Easystar platform

The Easystar platform is similar to the plane we are using for the task, but the existing platform is much lighter. The plane is however not self-stabilizing around the roll axis (X-axis).

The Easystar platform assumes measuring of levelness (`XY-sensor`) or/and the orientation of the aircraft while our platform uses a GPS only or a combination of GPS and IMU.

These differences should be kept in mind when adapting the software.

2.2 Sensors

We have used two sensor configurations available. One is with GPS only and the other is with both GPS and IMU.

2.2.1 GPS only

The only sensor we are using in this setup is a GPS for way-point navigation. Basically, the autopilot controls the plane directly without any stabilizing control loops in between. The GPS module is shown in Figure 2 and is mounted on top of the front of the airplane, as seen in Figure 1.

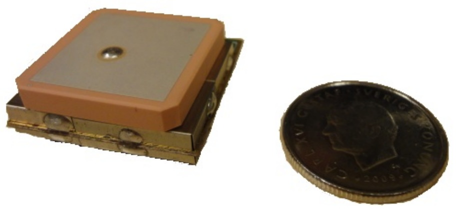


Figure 2: The GPS module that we use in the project. A 1SEK coin is also in the picture for size comparison.

2.2.2 GPS and IMU

In this setup we use both the GPS and the IMU for control of the airplane. The GPS and IMU measurements are used to make sure that the plane is heading in the right direction and e.g. measures what compensation that has to be made in pitch, when the plane turns sideways. The ArduIMU is described further in Section 2.4.

2.3 Autopilot

ArduPilot, [1], is an open source autopilot platform created by DIY Drones. The hardware consists of the core autopilot board which is sold by Sparkfun and various sensors and accessories that can be added for extra functionality. The board is displayed in Figure 3. This is the way it looks when it is shipped. To be able to use it the way it is intended to be used, we had to do some soldering of pins and a few cables, all of which is thoroughly described in [1]. Both hardware and the software are open source.

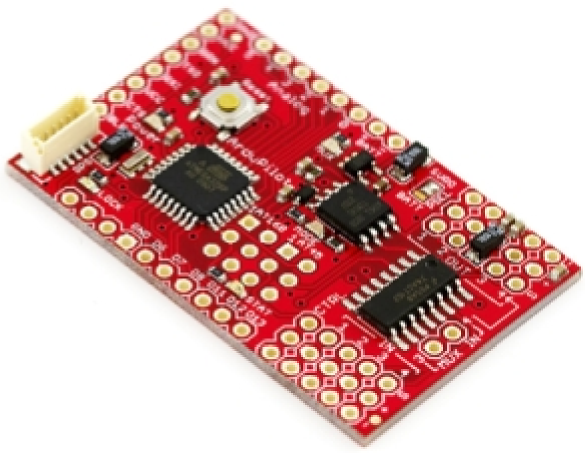


Figure 3: The ArduPilot board. Picture taken from [1]. To be able to use the board, several pins have to be soldered onto it. The white connection on the top left part of the board is used to connect the GPS.

2.3.1 Features

Some of the features are listed below:

- A GPS can be connected and 3D waypoints can be set by simply clicking in Google Maps.
- Controls elevator, rudder and throttle, plus an additional one such as an extra servo or ailerons.
- Thermopile XY and Z sensors or an IMU can be connected to enable measuring of the angles with respect to the ground and distance to the ground, respectively.
- Uses a cascaded PID control structure for stabilization and control of the airplane.
- Settings for the Easystar airplane are included which basically enable the user to get started right away with no setup (except maybe minor changes).
- The board can easily be (re)programmed by connecting the included USB cable to the ArduPilot board and the computer.
- The needed software can be downloaded from a homepage, where also the source code can be found. The source code can then be modified to your needs.

For a complete list of features, see [1].

2.3.2 Software

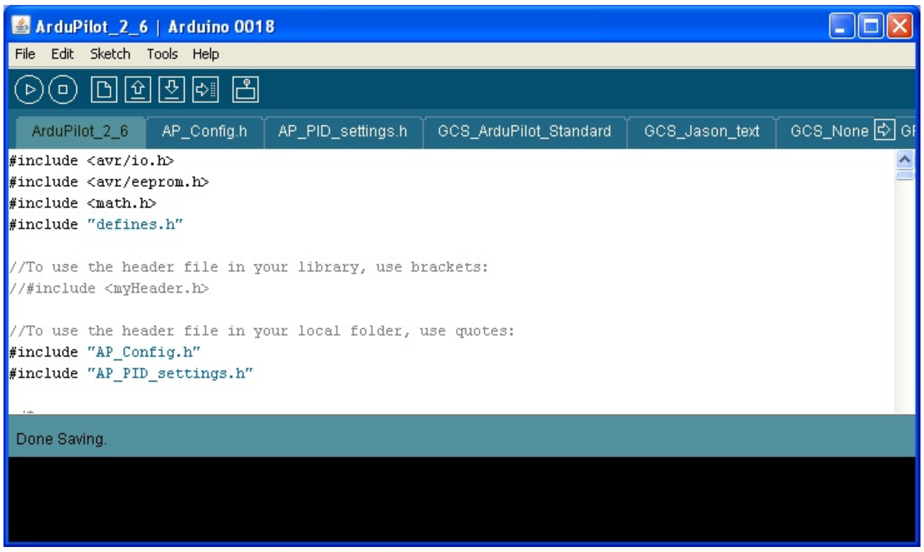
The source code to the Easystar airplane can be downloaded from the web, see [1]. There you can also download software to edit the source code and to compile and load it into the board using a USB-cable. This editor includes all of these features. A screenshot from the program is shown in Figure 4.

If you are going to use the Easystar plane, you can more or less load the software into the board right away, except for some configuration that you will have to do by changing some parameters in the C code header files. One file is for the configuration of the plane and sensors (`AP_Config.h`) and one is for configuration of the PID controllers (`AP_PID_settings.h`).

If you have several configurations for the same plane or different planes, you can use separate header files for the different configurations and just switch in the configuration you want.

2.4 ArduIMU

The ArduIMU board can be seen in Figure 5. Its software uses the Directional Cosine Matrix (DCM) algorithm to fuse the measurements from the gyroscope, accelerometer and GPS. Basically, the gyroscope measures the angular velocities which are integrated to obtain the orientation. This by itself gives a quite accurate orientation estimate, but it is subject to drift over time. The accelerometers are used to correct for drift by supplying a reference of the Z-axis of the plane and the GPS is supplying a reference for the plane X-axis, i.e. the heading angle



```
#include <avr/io.h>
#include <avr/eeprom.h>
#include <math.h>
#include "defines.h"

//To use the header file in your library, use brackets:
//#include <myHeader.h>

//To use the header file in your local folder, use quotes:
#include "AP_Config.h"
#include "AP_PID_settings.h"
```

```
Done Saving.
```

Figure 4: A picture of the Arduino editor where you can edit, compile and load the software into the board.

(which requires the plane to move to obtain a valid estimate). The centrifugal acceleration is compensated for by computing it as the cross product of the gyro vector and the velocity vector. The drift compensation is then implemented using a PI-controller. Additionally, normalization has to be made due to errors in the numerical integration calculations. More details about DCM can be found in [3].

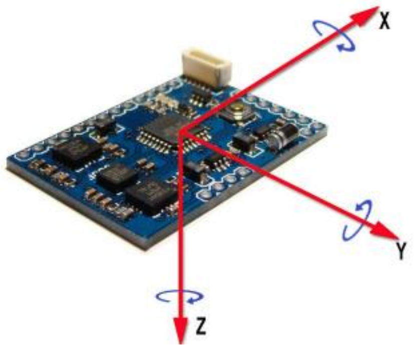


Figure 5: The picture shows the ArduIMU board. In order to use the board, some pins have to be soldered onto the board in order to be able to connect it to the ArduPilot. The picture is taken from [2].

2.4.1 Features

Some of the features of the IMU are listed below.

- 3 Axis Accelerometer.

- 3 Axis Gyroscope.
- The source code is Open Source.
- Prepared for use with ArduPilot.
- The same Arduino software can be used to edit, compile and load the software into the board as is used with ArduPilot.

For a complete list of features, see [2].

2.4.2 Software

The software can be downloaded from [2] and loaded into the board using the same USB-cable as when loading the software into the ArduPilot.

3 Software description

In this section of the report, we go through different aspects of the provided software with ArduPilot, with most relevance to our problem setup. First we briefly describe how the available program accomplishes its regular tasks. Then we discuss the existence of different flight modes and discuss their functionality, and finally we investigate the implications brought about by the setup of our existing platform and present a solution to overcome the existing implications.

However, before moving further to the subsequent sections, it is worth mentioning that there exist two header files within the provided software which are merely responsible for storing the values for different constants, which are used throughout the program, namely, `AP_config.h` and `AP_PID_settings.h`. As we later come across some of these constants, we point out within which of the mentioned files they are located.

3.1 Tasks

The software uses a modular structure to accomplish its predefined tasks such as data acquisition from available sensors, keeping stability of and navigating the UAV and contacting the actuators. Next we briefly go through these tasks.

3.1.1 Data acquisition

Within the problem formulation of the project we consider two different settings for the available plane platform, namely, the case with GPS as the only operating sensor and when an IMU is also available to assist GPS in navigating and stabilizing the plane.

1. **GPS:** Based on the type of the GPS, there are different communication protocols available in the software package. The GPS used in this platform requires use of *EM406* protocol, which can be set in `AP_config.h` header file. The main variables that contain the measured coordinates of the plane are
   - `current_loc.alt`
   - `current_loc.lat`

- `current_loc.lng`

which hold the values for the current altitude, latitude and longitude of the plane, respectively. These variables will be used for navigating the UAV among different checkpoints.

2. **IMU and GPS:** When using the ArduIMU in combination with a GPS, the GPS module is connected to the ArduIMU board instead of the ArduPilot board. As explained in Section 2.4 the GPS and accelerometer are used to eliminate the drift that occurs when the angular velocities are integrated to obtain roll, pitch and yaw angles. The variables that are being used in the ArduPilot software are

   - `roll_sensor`, i.e., the measured roll angle, which is used to calculate `servo_roll` and `servo_pitch`.
   - `pitch_sensor`, i.e., the measured pitch angle, which is used to calculate `servo_pitch`.
   - `ground_course`, i.e., the heading angle, which is used in the function `calc_nav_roll()` to calculate the `bearing_error`.

3.1.2 Navigation and Stabilization

In the provided software, the task of navigation and stability control are interconnected and are performed in a semi-simultaneous manner, i.e. neither are they performed at the same time, nor in a sequential manner. Considering the fact that it is possible to switch between different flight modes, as described in Section 3.2, the control structure and algorithm for the tasks at hand are different.

The main file responsible for coordinating navigation and stabilization of the plane is named after the folder that contains the software, in our case called `ArduPilot_2_6.pde`. There exists a function called `control_loop()` within this file, which in a nutshell implements three different loops with different speeds of manipulation, namely, fast, medium and slow. The fast loop runs with the frequency of 50 Hz. In this loop, first the new data is extracted from different sensors except the GPS. Then the *Flight mode* and *Take off* accomplishment is checked, and finally the stability of the plane is maintained based on a method depending on the current flight mode.

Among the tasks that are performed within the medium speed loop, the most important is to establish connection between the navigation and stabilization tasks. This is done by using a function called `calc_nac_roll()`, located in the file named `attitude.pde`. This function calculates a set point for the actuators to serve the navigation goal (which depending on the flight mode can be different), while maintaining the stability of the plane. This will be more clear when we discuss the control structure in Section 3.2.4.

And finally in the slow loop, the battery status is checked and data extracted from the GPS is updated. But above all the navigation task is taken care of using the function `navigate()`, which basically updates `nav_bearing`, i.e. the reference for the direction of the plane. It is worth mentioning that the reference direction is calculated within the X-Y plane, i.e.

$$\texttt{ref\_bearing} = a + b\arctan\left(-\frac{\texttt{current\_loc.lat} - \texttt{next\_wp.lat}}{\texttt{current\_loc.lng} - \texttt{next\_wp.lng}}\right), \qquad (1)$$

where a and b are merely constants that are set beforehand within the routine and `ref_bearing` is simply another name for `nav_bearing`.

3.1.3 Communication with actuators

The two actuators available in our platform are rudder and elevator. The variables that hold the desired angle on these actuators are `servo_roll` and `servo_pitch`, respectively. However, in order to communicate with the actuators we need the mentioned values in pulse widths. This is taken care using the functions within the file named `servos.pde`.

3.2 Flight modes

There exist several different flight modes already implemented within the software and you can also add any desired mode to the existing ones. Next we present some of the important already existing ones.

3.2.1 Manual

As it is obvious this flight mode requires full control from the ground station and while this mode is active the plane exactly follows the orders received from the ground station.

3.2.2 Fly by wire

This mode is mainly useful when you want to check the performance of the ArduPilot while you are still in the tuning phase, i.e. in this mode the plane navigates autonomously between the defined check points but you still have the power to modify the movement of the plane from the ground station.

3.2.3 Circle

While this flight mode is active the plane assumes a safe banking angle and rotates over the specified location (which may be current location).

3.2.4 Stabilize

This is one of the main flight/control modes available in the routine. This flight mode is responsible for stabilizing and navigating the plane between the specified way-points.

3.3 Control structure

In order to get a better understanding of the implemented algorithm, we proceed by describing the control structure implemented in the routine.

The control structure considered for stabilizing and navigating the plane is a cascaded structure. Figure 6 and Figure 7 illustrate the control structures for stabilizing and navigation concerning pitch and banking dynamics, respectively.

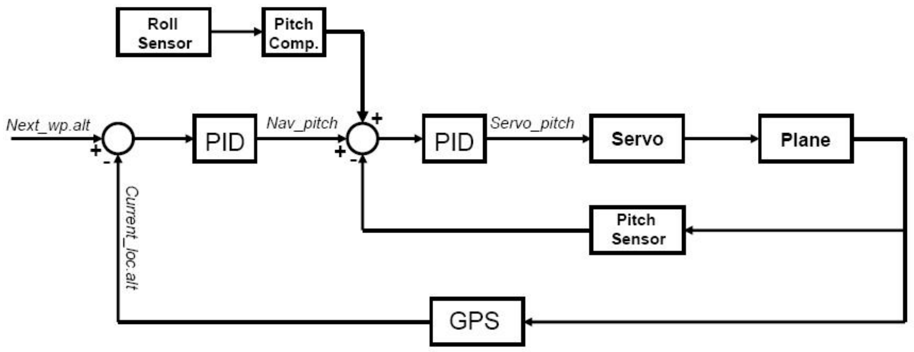


Figure 6: The figure shows a block diagram of the pitch angle control structure.

3.3.1 Pitch control

We start with Figure 6, and discuss how different variables involved are calculated. Starting from the outer loop, `next_wp.alt` is a predefined value which specifies the altitude of the next checkpoint from the list of defined checkpoints. As was mentioned in Section 3.1.1, `current_loc.alt` is the altitude of the current location which is extracted from the GPS reading. Having defined these quantities, the altitude error is considered as the input to a PID controller which results in the variable `nav_pitch`. This value represents the required pitch angle of the aircraft in order to reach the altitude of the next way-point. The inner loop will then use this quantity to produce this required pitch angle. This task is done via utilizing a PID angle, the output of which will be the required elevator angle to produce the required pitch.

However, as can be seen from Figure 6, this is not the only task that the inner loop should accomplish. This is due to the fact that, whenever we are starting to turn if this is done with zero elevator angle we will also start to lose altitude, as a result there exist another factor added to the set point of the inner loop, which compensates for the loss of altitude in these scenarios.

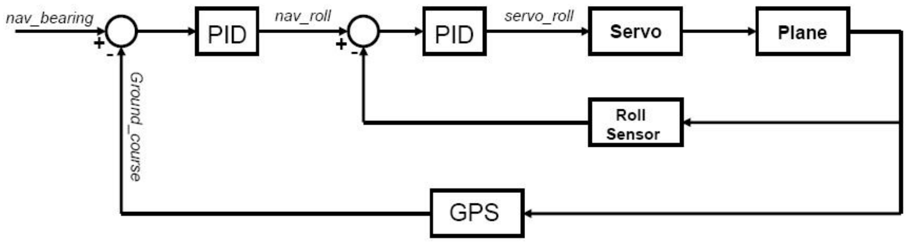


Figure 7: The figure shows a block diagram of the banking control structure used in the ArduPilot software.

3.3.2 Banking control

Figure 7 illustrates how the program keeps the stability with respect to roll dynamics, and navigates the plane concerning the direction or bearing of the plane. If we start from the outer loop, `nav_bearing` which serves as the set point, represents the correct direction that the plane should be facing in order to get to the next way-point. Then using the estimated `ground_course` from the GPS, the bearing error

$$\texttt{bearing\_error} = \texttt{nav\_bearing} - \texttt{ground\_course} \tag{2}$$

is calculated. This is further used (through the outer loop PID controller) to generate the required banking to reduce the bearing error. This variable is denoted as `nav_roll` and serves as the set point for the inner loop. The task of the inner loop is then only to produce the correct rudder angle for the plane to follow the required banking behavior (`nav_roll`). The required rudder angle is denoted `servo_roll`.

It is worth mentioning that, although the function `stabilize()` is run within the fast loop, the bearing error and altitude error are generated within the medium and slow loops.

4 Software modifications

Since the Easystar platform assumes that the orientation is measured using XY-sensors or IMU, some modifications have to be made in the control structure when this this is not the case. In this section we describe what parts that have been modified and what has been changed.

4.1 Implications

One of the platforms considered within this project is only equipped with the GPS. By examining Figures 6 and 7 closely, it becomes apparent that the missing pitch and roll sensors can harshly temper with the stabilization and navigation performance of the control structure. Next we discuss the implications brought about due to lack of these two sensors.

Without the mentioned two sensors, first the inner loop of the two control structures disappear. This deteriorates the performance of the control structures. However in case of the pitch control, the situation becomes more dire. As you might recall, one of the responsibilities of the pitch control is to compensate for the loss of altitude due to banking of the plane. However, without the roll sensor data and due to the slow feedback from the GPS (not very accurate measurements of the altitude), delivering this goal becomes impossible using the existing control structure. This causes severe problems during maneuvers which include banking.

In the following section, we propose possible solutions to these implications.

4.2 Solution

The solution to the problem with the inner loops can be solved by simply omitting those loops from the control structure and use a simple feedback with a single PID controller. The deterioration of the performance can be alleviated, to some extent, by conducting more rigorous tuning and/or increasing the speed of GPS reading, making the control loop faster than the previous structure.

Although the mentioned modified control structure solves the issues with the inner loops, it fails to compensate for altitude drop while banking. One of the possible ways to deal with this issue is to explicitly define a relationship between the rudder angle (`servo_roll`) and the elevator angle (`servo_pitch`), such that while issuing a banking (with its respective rudder angle) when traveling with constant throttle, the altitude remains constant. This can be achieved by identifying this relation or defining a look-up table consisting of different values of elevator angles versus different speeds and rudder angles.

In order to obtain these valid data points, i.e. perfect compensation for altitude loss, series of experiments are required to scan through different elevator angles while keeping the speed and rudder angle constant. Having achieved a valid point, we continue with changing speed or rudder angle one at a time and repeating the same procedure. This will eventually result in a look-up table or in case of enough available data points, brings the possibility of actually identifying the relation between elevator angle and the remaining two variables.

Figures 8 and 9 summarize the proposed solutions to lack of pitch and roll sensors.

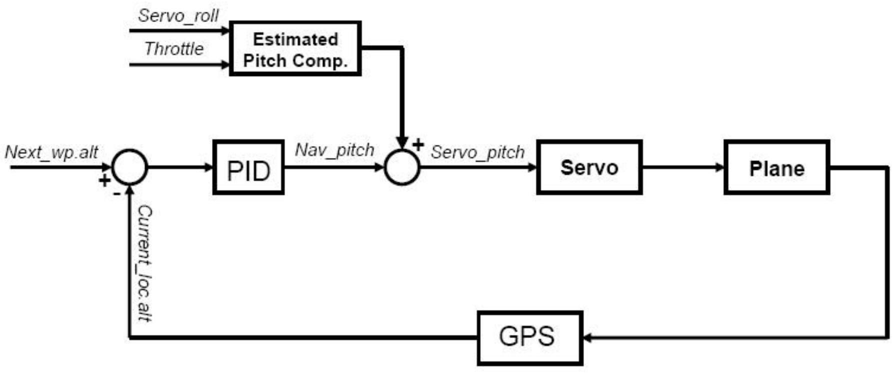


Figure 8: The figure shows a block diagram of the modified pitch control structure used in the ArduPilot software.

As can be seen from Figure 8, the pitch compensation works as an offset to the required elevator angle to navigate to the altitude of next way-point. It goes without saying that, *Estimated pitch comp.* block represents either the obtained look-up table or the identified relation.

5 Achievements and conclusions

As described in Section 1, we set up some goals for the project before we started working on it.

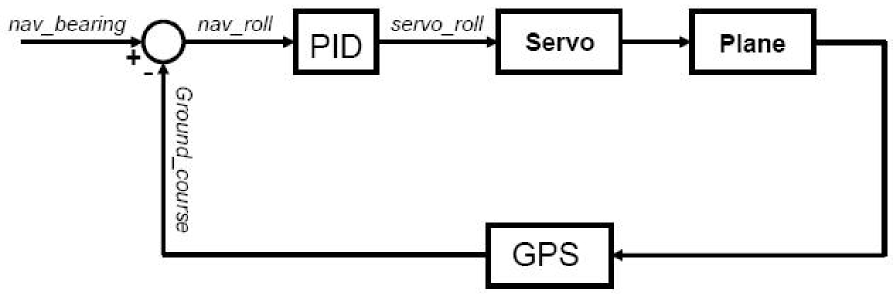


Figure 9: The figure shows a block diagram of the modified banking control structure used in the ArduPilot software.

5.1 Goals

Within this project, so far we have matched the existing platform with the provided software which was compatible with the Easystar platform. On the way of accomplishing this task we have pointed out issues that might rise due to this task and also provided practical and feasible solutions for the mentioned implications. However considering the light structure of the available platform and unstable weather conditions, we have not been able to finish the experients.

However, in case we get the chance to carry out the mentioned experiments, setting up way-points and navigating among them is already implemented in the software package and it does not require any modifications.

5.2 Extra goals

We have connected an IMU as an extra sensor, but due to the weather conditions we have not yet been able to carry out any experiments using this configuration.

6 Exercises

1. Setting up the hardware.
   (a) Configure the header file `AP_config.h` according to the available hardware by using the Arduino editor. Use the SIRF protocol for the GPS.
   (b) Upload the code to the Ardupilot board.
   (c) Connect the GPS and confirm that the data extracted from the GPS is valid by reading from the serial output from the Arduino editor.
   (d) Configure pde-file for the Ardu IMU (`arduimu.pde`) according to the available hardware. Use the EM604 protocol for the GPS.
   (e) Connect the IMU and check that the data extracted from the IMU is compatible with the movement of the board. Hint: You can use either the serial output or Ardu IMU test program (which includes an animation).
2. Setting up the software.

(a) Find which three control modes that are currently used in the software.

(b) Make sure that manual mode is one of the available modes. Compile and upload the software into the Ardupilot.

(c) Control the actuators using the tranceiver.

(d) Modify the control structure to write your own values so that you pre-specify the angles of the control surfaces to be +10 degrees.

3. Control Structure

   (a) Find out how the software works in the "stabilize" control mode.

   (b) Modify the existing control structure to comply with the GPS only setup as illustrated in Figure 8 and Figure 9.

References

[1] “ArduPilot”, [Online; extracted September 13, 2010], `http://code.google.com/p/ardupilot/wiki/ArduPilot`

[2] “ArduIMU”, [Online; extracted October 1, 2010], `http://code.google.com/p/ardu-imu/`

[3] William Premmerlani and Paul Bizard, ”Direction Cosine Matrix IMU: Theory”. [Online; extracted October 4, 2010], `http://gentlenav.googlecode.com/files/DCMDraft2.pdf`

Avdelning, Institution
Division, Department

Division of Automatic Control
Department of Electrical Engineering

Datum
Date

2011-03-02

Språk
Language

☐ Svenska/Swedish

☒ Engelska/English

☐ ________________

Rapporttyp
Report category

☐ Licentiatavhandling

☐ Examensarbete

☐ C-uppsats

☐ D-uppsats

☒ Övrig rapport

☐ ________________

ISBN
—

ISRN
—

Serietitel och serienummer
Title of series, numbering

ISSN
1400-3902

LiTH-ISY-R-2996

URL för elektronisk version

`http://www.control.isy.liu.se`

Titel
Title
Project report - ArduPilot

Författare
Author
Sina Khoshfetrat Pakazad, Daniel Ankelhed

Sammanfattning
Abstract

Ardu software, for autopilot functionality has been used for several years for many different UAV platforms. The aim of this project is to use the same functionality for the plane platform existing in the department. Considering the mismatch of the existing platform with the previously used ones for Ardu autopilots it is of great importance to alter the software and hardware provided by Ardu in order to be able to fly the existing UAV.

Nyckelord
Keywords
UAV, autopilot, IMU, ArduPilot, ArduIMU.